W3C mobileOK Basic Tests 1.0 Checker (Beta Release) Developer Manual

Version of this document: 0.5 (Draft)

Date of Release: 2008-02-04

Editors: Abel Rionda, Ignacio Marin (Fundación CTIC)

Contents

Illustration Index

1 Introduction

1.1 Purpose and Audience

This document is meant to serve as a Developer Manual for the W3C mobileOK Basic Tests 1.0 Checker (Beta Release). Current version of this Developer Manual is a draft, so the editors intend to write new versions correcting typos and errors and inserting new and more complete and accurate information.

The version of the mobileOK Checker subject to comments in this document is the second public release of the checker (a Beta version after the first Alpha version). A final version of this software, of this User Manual and of the Developer Manual are expected later in 2008.

The audience of this document are developers being aware of the need to check conformance of web resources (and also of the way in which they are delivered) against W3C mobileOK Basic Tests 1.0 and the need to implement conformance checking tools based on those tests. The W3C mobileOK Checker is a software tool, created by means of collaboration among different organizations (see Acknowledgements). The intention of such Task Force is to create a Reference Implementation so the W3C mobileOK Basic Tests 1.0 document can become a W3C Recommendation. This checker allows users to pass mobileOK Tests on a given web resource.

W3C mobileOK Basic tests implement the way in which some of the Best Practices in the W3C Mobile Web Best Practices (Basic Guidelines) 1.0 document can be checked against web resources. The term “some” refers to all those Best Practices that can be verified by a machine (i.e., which can be described as an algorythm). Those Best Practices which can not be described as an algorythm (needing the assessment of a human expert) are covered by the future W3C Mobile OK Pro Tests document, still in development.

This document is accompanied by the W3C mobileOK Basic Tests 1.0 Checker User Manual. While this Developer Manual describes the internal architecture, design and implementation details of the W3C mobileOK Checker software tool as it would be interesting for developers willing to modify, extend or enhance the functionality of the tool or even willing to integrate it with their own software or services the User Manual tries to explain the W3C mobileOK Checker as a black box and to cover all the subjects of interest to anyone trying to use the tool as-is.

The W3C mobileOK Checker 1.0b, its User Manual and Developer Manual, the W3C mobileOK Tests 1.0 and the W3C Mobile Web Best Practices (Basic Guidelines) 1.0 are efforts taking place under the framework of the W3C Mobile Web Best Practices working group (part of the W3C Mobile Web Initiative).

1.2 Copyright Issues

The W3C mobileOK Basic Tests 1.0 Checker software tool software tool and related documents (User Manual and Developer Manual) are released under the W3C License, in its current version at the very moment of the release of this document.

Several external Java libraries and software tools are used by the W3C mobileOK Basic Tests 1.0 Checker. Their use is subject to different licenses and copyright clauses. The binary distribution for the W3C mobileOK Checker and its source code in W3C's CVS repository contains the licenses and copyright claims to which each of them are subjected.

Eclipse, the IDE used as example in this documentation, is released under the Eclipse Public License v1.0 in the version used for the screenshots taken.

1.3 Structure

This document is divided into several sections, which are now briefly described:

- Introduction: Current section, where the general purpose of this document and of the target software tool (W3C mobileOK Basic Tests 1.0 Checker) is commented, including related W3C activities and documents.
- Download and installation: This section describes how to download the Java executable binary file (in JAR format) for the W3C mobileOK Checker, how to optionally download the source files for the software and the library dependencies for the Checker.
- Software Requirements (Use Cases): Description of software requirements for the W3C mobileOK Checker.
- Design: This section explains the software design using UML artifacts. In addition, other aspects of the development, such as the unit testing, are introduced.
- Cookbook: Several examples to illustrate how to use the checker from Java code.
- Extending the library: Implementation decisions which should be taken into account by someone who is interested into extending the library.
- Identified improvements and extensions: Issues regarding future improvements of the Checker.
- Known issues: Issues regarding known bugs and future improvements of the Checker, and missing information (that will be included in future versions) in this Developer Manual.
- References: Documents interesting for readers, which might help them to understand this document, the W3C mobileOK Checker 1.0b software tool and all related work of the W3C Mobile Web Best Practices working group.
- Acknowledgements: Credits to the people who helped developing this Developer Manual and the W3C mobileOK Basic Tests 1.0 Checker software tool.

2 Download and installation

2.1 Binary archive download

The Java binary executable JAR file is downloadable at:

http://dev.w3.org/cvsweb/2007/mobileok-ref/

The existing version of the W3C mobileOK Basic Tests 1.0 Checker when releasing this document is available at:

http://dev.w3.org/cvsweb/~checkout~/2007/mobileok-ref/mobileOK-Basic-RI-1.0-deploy.jar?rev=1.8&content-type=text/plain&only_with_tag=HEAD

2.2 Source code download (from CVS repository)

The source files corresponding to the JAR file mentioned in 2.1 are available in W3C's CVS repository at :pserver:anonymous@dev.w3.org/public/sources/2007/mobileok-ref (password for anonymous user is *anonymous*). If the previous URL does not work, the specific data for CVS access are commented below:

- Host: dev.w3.org
- Repository path: /sources/public
- User: anonymous
- Password: anonymous
- Connection type: pserver
- Port: default pserver port (2401)
- Specified module name: 2007/mobileok-ref.
- Available branches: Only HEAD. No other branches expected in the future.

Please take into account that, in order to be able to deal with the source code as a developer, you might need to read this W3C mobileOK Basic Tests 1.0 Checker Developer Manual.

3 Software Requirements (Use Cases)

This section is based on the Software Requirements set for the W3C mobileOK Basic Tests 1.0 Checker as defined in the thread of the mailing list of the Task Force in charge of the development of the tool (http://lists.w3.org/Archives/Public/public-mobileok-checker/2007Mar/0009.html).

3.1 Licensing

The output of the group's work is to be open source and is to be freely licensed under the terms of license specified in the Copyright Issues section of this document.

3.2 Conformance

1. The output of the reference checker MUST state which version of mobileOK is being checked against and MUST state its own version number.
2. The checker assesses conformance with a number of standards and recommendations other than mobileOK (external standards).
3. Where possible the checker SHOULD NOT create independent implementations of external standards conformance checkers and should use accredited, where possible, or de facto standard implementations.

3.3 Deployment/Use Cases

It is the intention that the checker will be a component of the following deployments:

1. As a component of a public service where users type a URI for checking into a user interface.
2. As a component of a public service which offers a programmatic interface for use remotely by other programs.
3. As part of an IDE.

3.4 Structure

1. The checker will have a manifest architecture that is documented separately from its code.
2. Input to the checker will be specified by URI.
3. Other inputs to include varying the request headers and request partial execution of the tests.
4. The checker will be written in Java.
5. The checker development project will not develop a user interface except as necessary for testing it, but the use case of its deployment in a human request / response environment should be borne in mind. Specifically this should not be seen as a project to create the W3C mobileOK checker.
6. The checker will create an intermediate document that makes available for inspection all details of retrievals, validation and other pre-processing required in order to carry out the tests. The format of this intermediate document will be specified separately, and will use existing representations [like RDF/HTTP] where possible.
    1. To take account of the possibility of input documents that are not well-formed or that may be invalid (or may just be binary), the checker will embed them in the intermediate document in a form that preserves the integrity of the input document while making it possible to reconstruct the original document.
    2. To allow processing of mal-formed and invalid primary input documents (those that are the subject of the test, rather than resources that are referenced) the pre-processing will provide a 'cleaned up' version and that the nature of the clean-up needs to be explicit and not implementation dependent.
    3. The intermediate document is to be suitable for audit purposes and MUST contain sufficient information to justify test results.

4. It will contain the external resources required to complete the tests together with their retrieval information and validity information.
5. HTTP parameters and their values should be recorded in a normalised form as well as being recorded in their original form.

7. The checker will produce a test report document describing the results of carrying out the tests together with appropriate WARNs and other diagnostic information. The format of the result document will be specified separately and will use appropriate existing representations.
    1. It must be possible to request the inclusion of the intermediate document as part of the report.

3.5 Process

1. Source code and documentation for the project will be kept at W3C CVS repository, in a project called W3C CVS Repository for mobileOK Checker.

4 Design

4.1 High Level Design

As it was mentioned, the checker is based on having a first phase of preprocessing or building an intermediate document and a second phase of executing XSLT based tests against that intemediate document. Because of that, main design classes can be clustered in these two phases.

Illustration 1 shows a high level architecture diagram. Note how the relationship between the input and output of the library, and the library itself, are fulfilled.

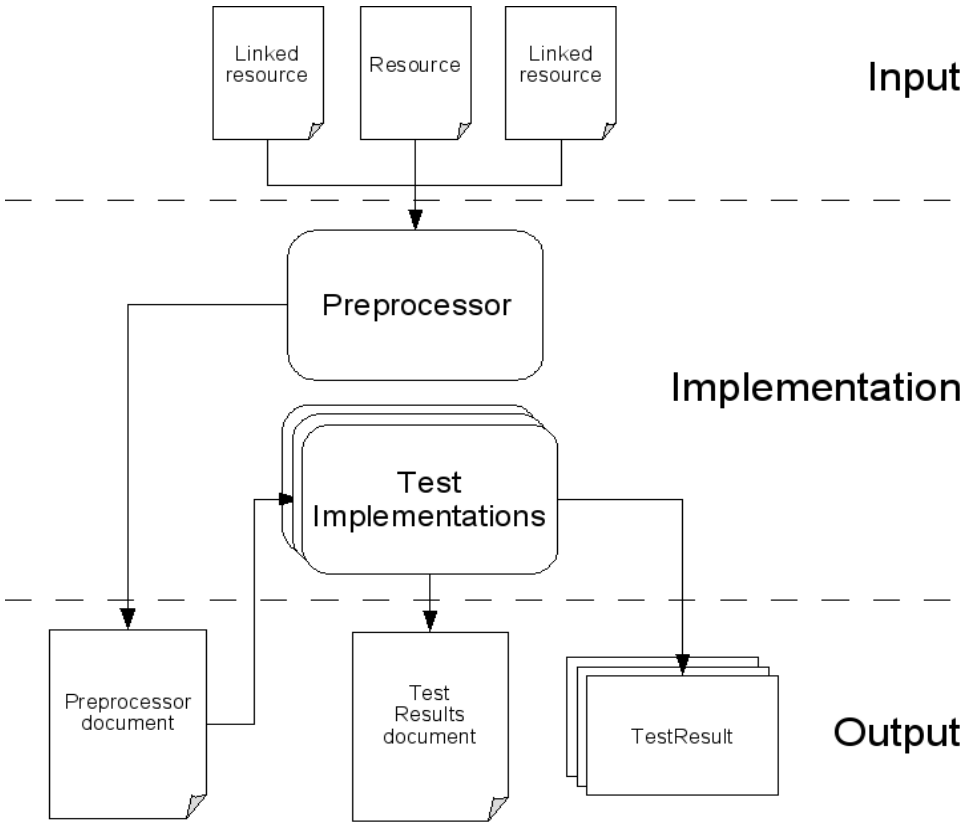

Illustration 1: High Level Design

4.1.1 Input

The input to the library is a main resource and others linked resources that are referenced from inside this main resource (e.g. images, stylesheets). In addition, these resources can have another ones embedded inside them (e.g a stylesheet that has an import directive inside).

4.1.2 Implementation

Preprocessor

The classes involved in this phase are in charged of building the intermediate document (a.k.a moki). For this purpose, when the input is received, several objects are instantiated, mainly the ones that represent the main document istself, images, objects, CSS and others linked resources.

Once these objects are built, the preprocessor phase finishes with the construction of the intermediate document which structure can be found in the mobileOK Checker 1.0b User Manual document. Note that this document has the sufficient information that enables the execution of the test in a declarative way.

This intermediate document is the input for the next phase and it is one of the documents that is returned as the output.

Test Implementation

Once the intermediate document is built, XSLT based tests are launched against this document using XPath expressions. It is important to note that all tests are XSLT based except the ones related to CSS. Currently, CSS tests are not declarative because CSS stuff is not serialized in the intermediate document. This can be a future improvement (see Representation of CSS in XML) and so far there is a first prototype of this serialization in the CVS repository.

4.1.3 Output

The library produces, in addition to the intermediate document, a results document (which format is presented in the mobileOK Checker 1.0b User Manual document) and a set of Test Result objects with extra information.

4.2 Main classes involved in preprocessing phase

4.2.1 Static View

The key classes involved in preprocessing tasks can be identified in the class diagram in Illustration 2.

4.2.1.1 Tester class

This is the main class and the entry point to the library. A URI or a file is neccesary for the library to be executed. This class has two essential attributes in this phase: Preproccesor and PreprocessorResults.

4.2.1.2 Preproccesor class

This class is responsible for parsing the incoming resource and for creating the objects that can represent such resource (main document, CSS, Images, and so on). These objects are kept by ProcessorResults.

4.2.1.3 PreprocessorResults class

This class has all the necessary to build the intermediate document. This class keeps some other classes to model HTTP resources, such as:

- **HTTPXHTMLResource**: This class represents the main resource retrieved and has several attributes such as the DOM tree (and a tidied version), information about validation of the markup and inline or embedded CSS (i.e. style attribute or style tag) contained inside it. In general, there is a complete hierarchy starting from a basic HTTPResource (managing HTTP connections, redirections and so on) to specific classes such as HTTPXHTMLResource or HTTPImageResource. A common attibute for all these HTTP classes is a URI.

- **CSSResource**: This is the common interface for linked CSS resources (modelled as HTTPCSSResource class), embedded CSS resources (EmbeddedCSSResource class) or inline CSS resource (InlineCSSResource class). All these classes have information about the validity of CSS (implemented using CSS Validator) and the content of CSS documents themselves. One important issue is related to the section 2.4.8 of the W3C mobileOK Basic Tests 1.0 document [http://www.w3.org/TR/mobileOK-basic10-tests/#validity]. Basically, a CSS resource is considered valid if it validates against CSS Level 1 grammar. On the other hand, the presence of at-rules, properties or values or combinations of properties and values that are not specified in CSS Level 1 does not constitute a validity failure for CSS. In this context, it is needed to register both raw CSS and some kind of proccessed CSS (without estraneous attributes) for the sake of validating against CSS Level 1. For this purpose all CSS implementation classes have to take account of this.

- **HTTPImageResource**: It represents an image resource having attributes such as dimensions, validation errors, and so on.

- **HTTPObjectResource**: Represents an object resource with attributes such as content-type.

- **MokiDocument**: It contains a DOM tree with the intermediate document or moki. This document is one of the output of the library.

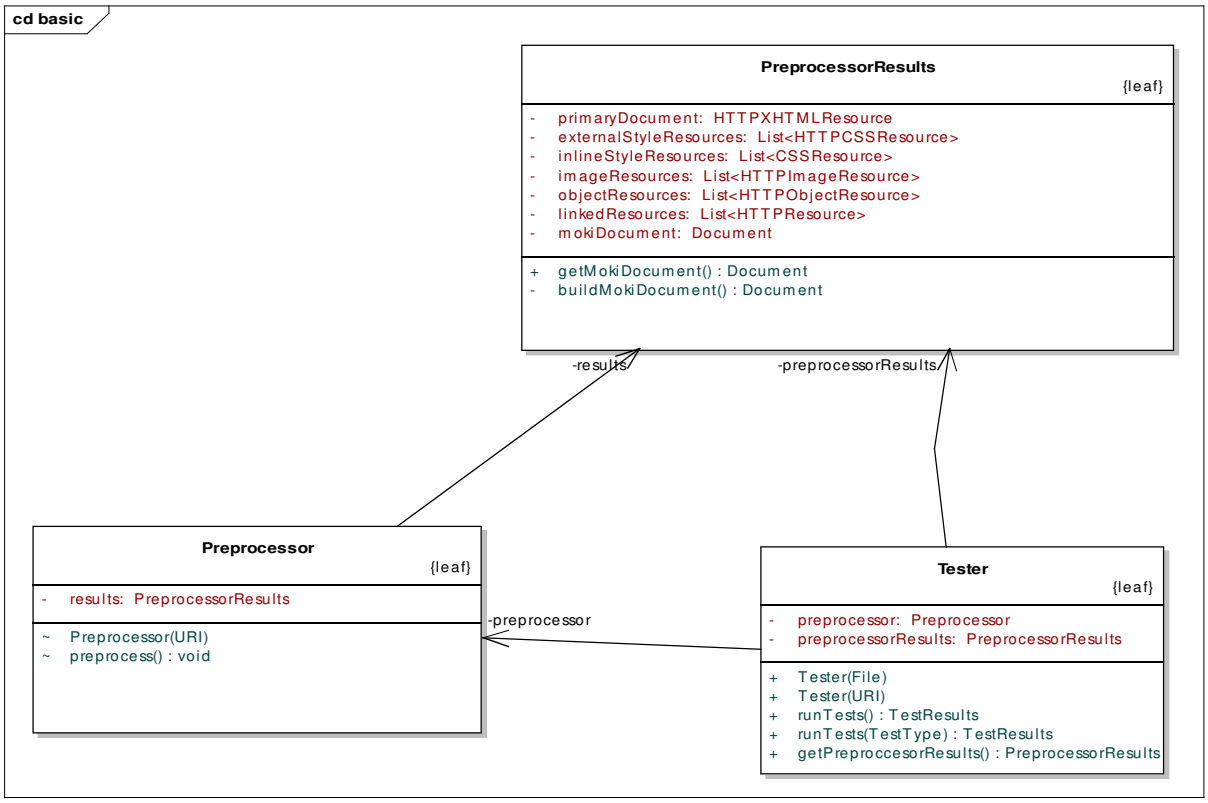

Illustration 2: Preprocessing phase overview

4.2.2 Dynamic View

In Illustration 3, the sequence for creation of the objects representing the whole Web resource is shown. This is triggered through the getPreprocessorResults() method of the Tester class. First of all, an object representing the main document is created (HTTPXHTMLResource).This object provides lists of URIs for images, CSS or links. For each of these lists, the Preprocessor object builds specific ones for these elements (HTTPImageResource, HTTPCSSResource and so on). In some cases it is necessary a further treatment of the information contained in these elements (e.g CSS External , CSS Embedded or Images. See the note in the Diagram at Illustration 3 for more information on this).

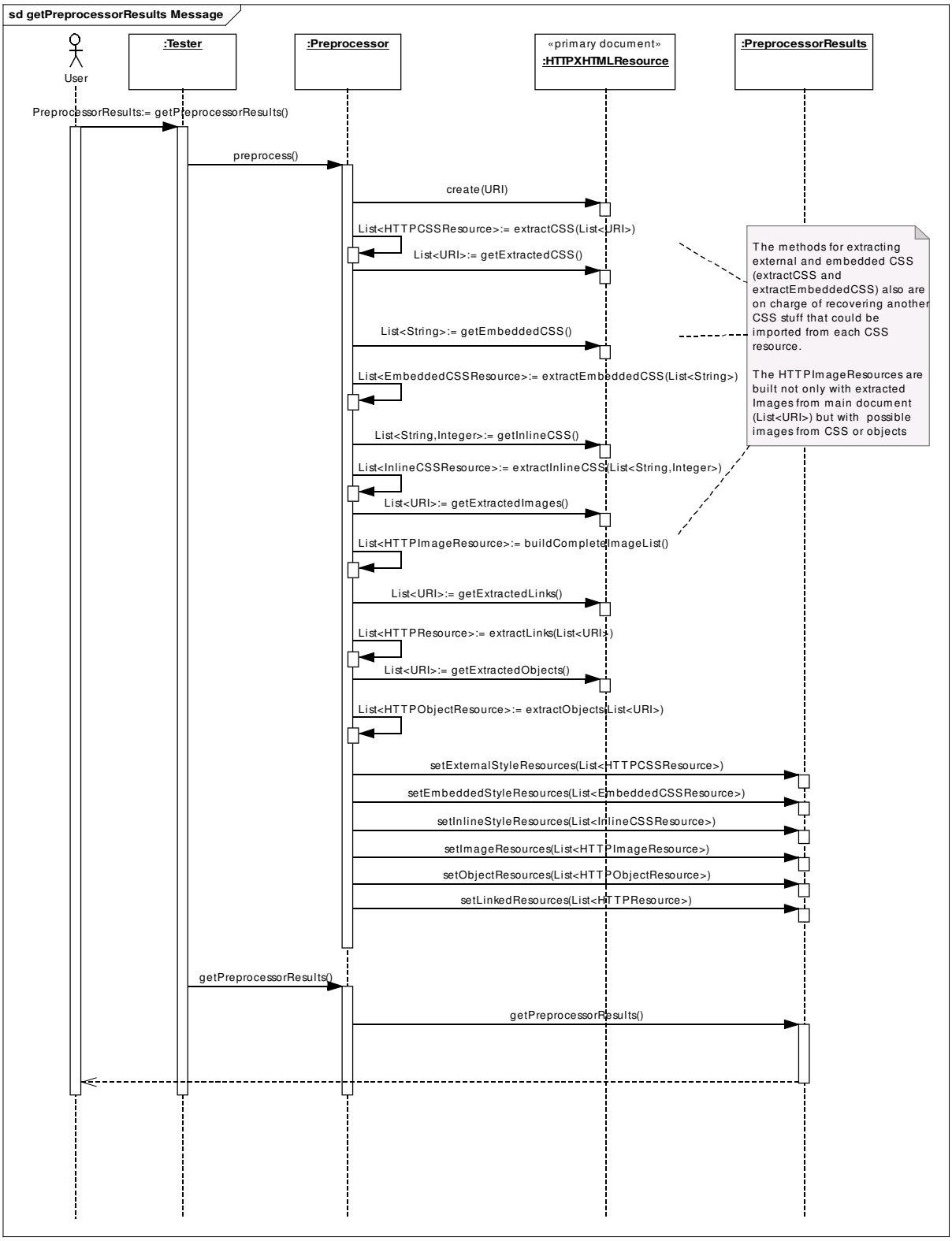

Illustration 3: Sequence Diagram for the getPreprocessorResults method

Another possible method is getMokiDocument(). This method (that internally uses the previous one -getPreprocessorResults()-) is to build the intermediate document. Note that the method on charged of doing that task is, at the end, buildMokiDocument() of PreprocessorResults class.

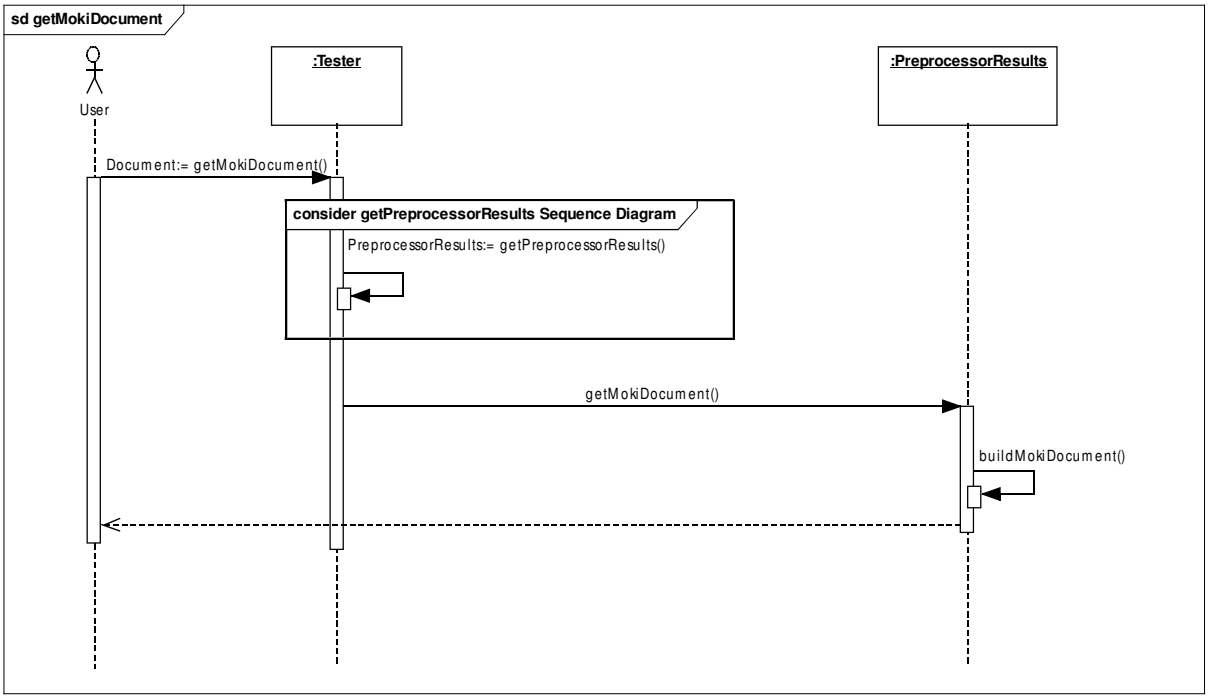

Illustration 4: Sequence Diagram for the getMokiDocument method

4.3 Main classes involved in test implementation phase

The classes involved in this phase are the ones related to the implementation of mobileOK Basic 1.0 Tests and to the results generated by these tests.

4.3.1 Static View

4.3.1.1 Results of the library

Illustration 5 Represents the main classes related to the generation of the results output by the checker.

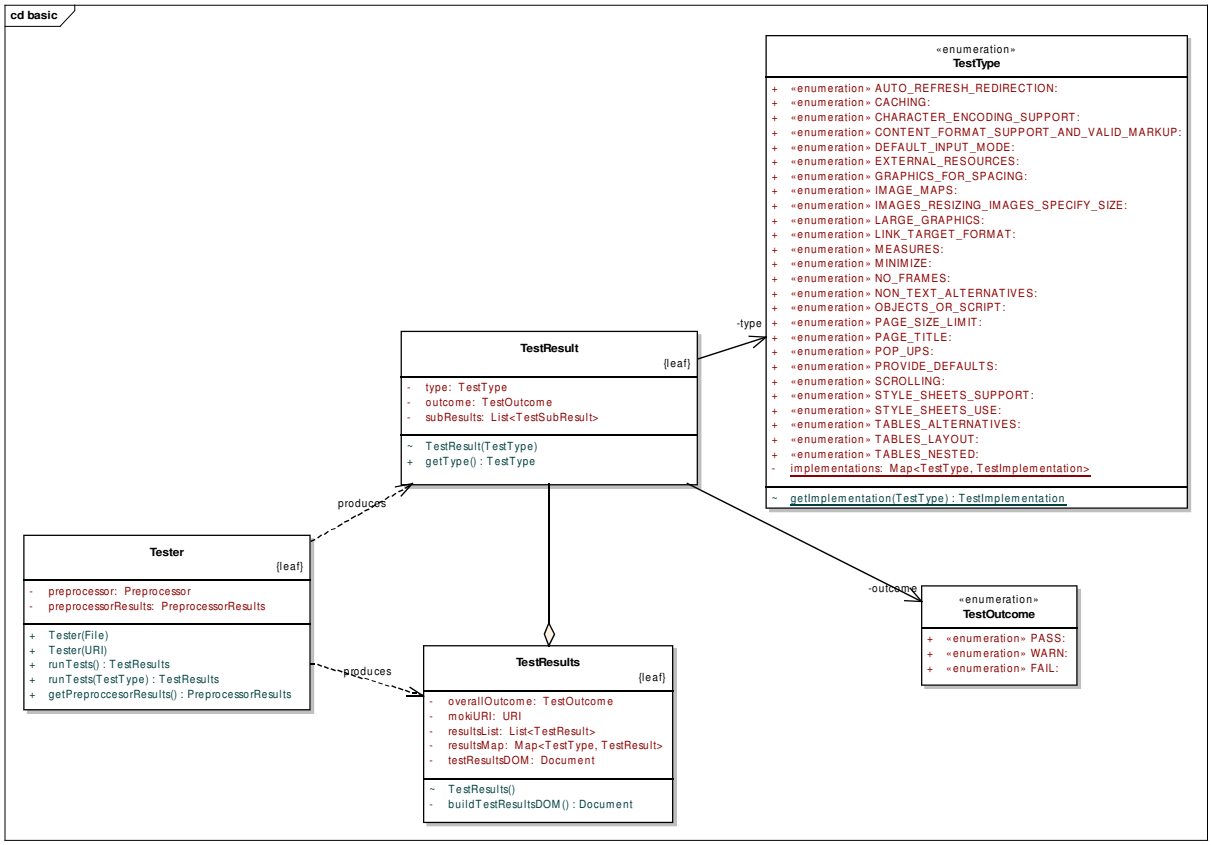

Illustration 5: Main classes involved in Test Implementation phase

- **Tester**
  The main class in the library exposes two methods for launching the tests. It exposes a `runTests()` method that runs all the tests, or, `runTest()` which runs just one test. Both methods return TestsResults/TestResult respectively containing the output of the tests. These objects are used for building the result document.

- **TestType**
  It is an enumeration of all test types.

- **TestResults/TestResult**
  Returned by `runTests()` and `runTest()` respectively, they encapsulate the results of many tests and one test, respectively. TestResults includes many TestResult instances. These objects include test outcome, warnings, and information.

- **TestOutcome**
  It is an enumeration of the the two tests outcomes, `PASS` and `FAIL`. On the other hand, one test with PASS outcome can have additional WARN outcomes.

- **TestException**
  This is the exception that is thrown if something goes wrong in the tester. This can be both something internal to the library as an external problem (for example a problem accesing the input document).

4.3.1.2 Implementation of the tests

In the following diagram (Illustration 6), the main classes regarding tests implementation can be observed.

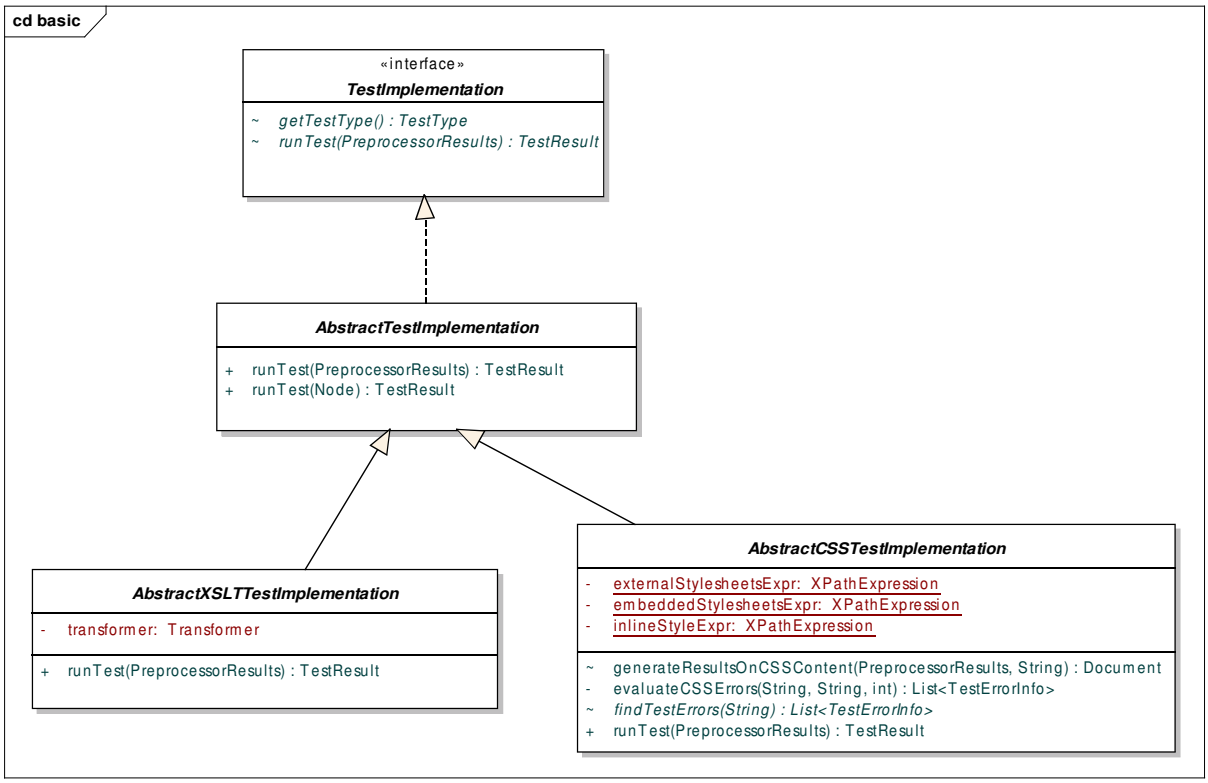

Illustration 6: Test Implementation Inheritance

- **TestImplementation**

  This is the top interface for all mobileOK tests. The interface method is `runTest()`. This method runs on the intemediate document object (the output of preprocessing phase) and produces a TestResult object.

- **AbstractTestImplementation**

  This class defines one method that should be implemented in the derived classes [TestResult runTest(PreprocessorResults)]. Additionaly, it provides a utility method (used by derived classes) that builds a TestResult object from a DOM node. [TestResult runTest(Node)]

- **AbstractXSLTTestImplementation**

  This abstract class has the responsability of applying the XSL transformations against the intermediate document and producing a TestResul object. The stylesheet applied depends on the specific test implementation that calls this method.

  This is the approach for most tests unless CSS related ones. For example, in the next diagram two concrete test implementations as Caching and TablesNested are showed.

  Note that some helper methods in order to fulfill the test are included in it (as it can be seen in the Caching Test)

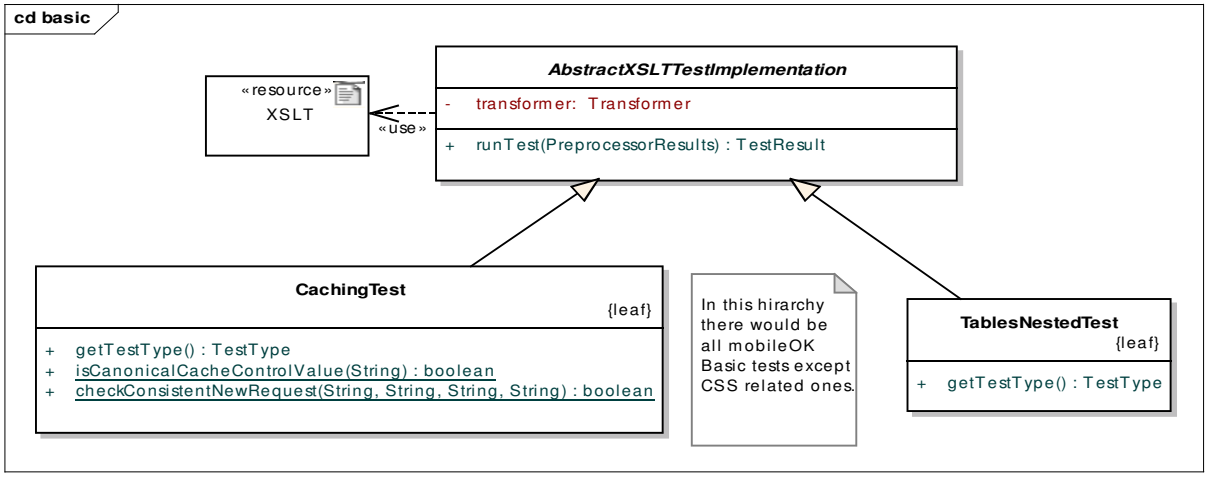

Illustration 7: XSLT Test implementation

- **AbstractCSSTestImplementation**

  As it was said, currently CSS tests do not follow a declarative approach because CSS stuff is not serialized in the intermediate document. Except CSS stuff embedded in the markup of the main XHTML, all the CSS is kept in moki in a raw format (External or imported stylesheet). Because of that, CSS tests are managed in a different way.

  First of all, this class has several XPath expressions in order to access the parts of the moki where external, embedded or inline CSS are present. Then, each CSS piece is validated against the current CSS test implementation, through a regular expression. The output of these regular expressions is used to build the Test Result object.

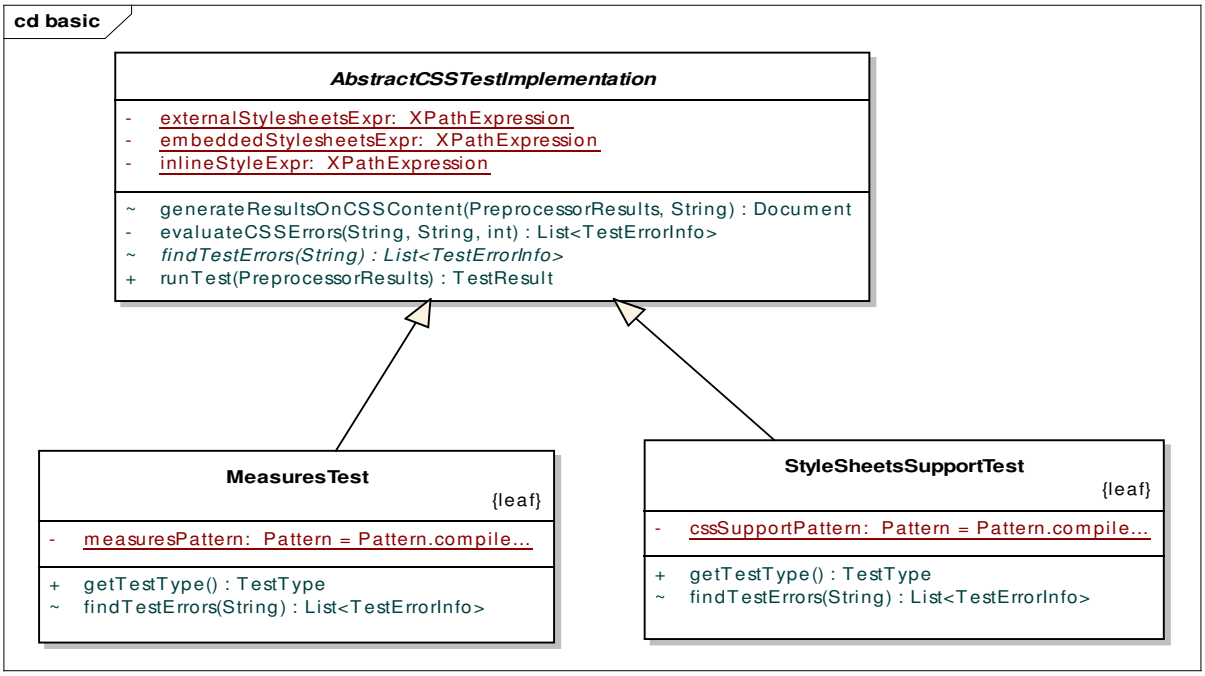

Illustration 8: CSS Test Implementation

4.3.2 Dynamic View

Illustration 9 represents the dynamic diagram for the runTests() method from Tester class. Note that, for each test received in an array of TestType, the following steps are taken:

- First of all, it is obtained the implentation class for the specific TestType. This class will be a child from either AbstractXSLTTestImplementation or AbstractCSSTestImplementation.
- Then, the method runTest() from the implemtation class is launched.
- The result (of TestResult type) is saved in an array of TestsResults.
- After the loop, the TestResults array is returned.

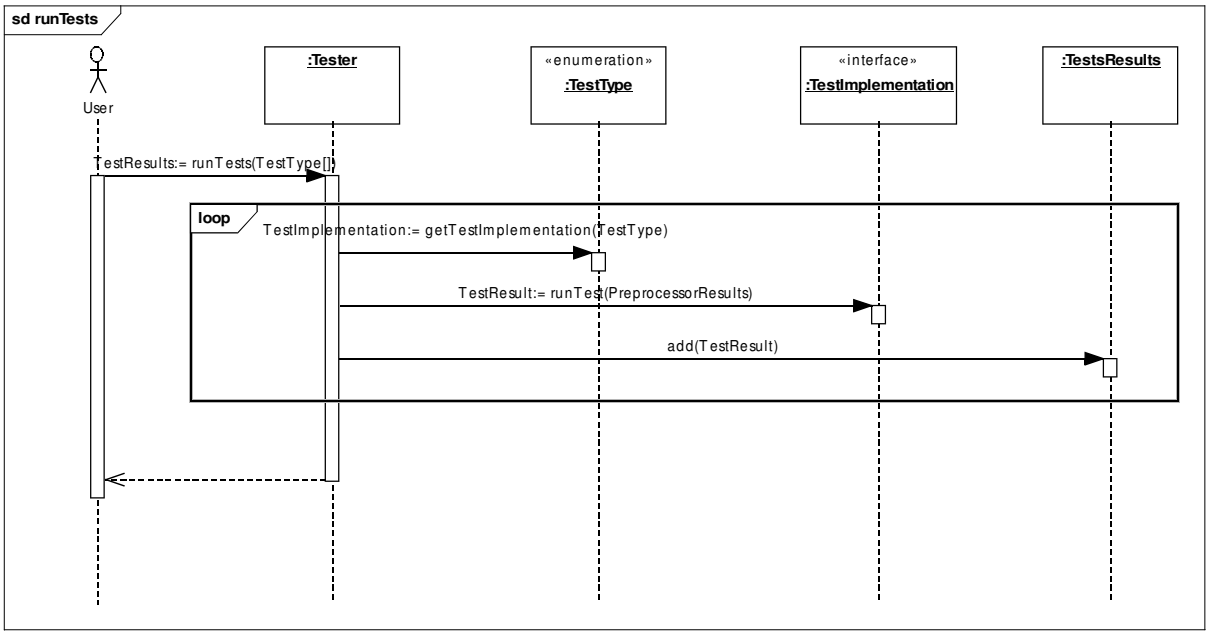

Illustration 9: Sequence Diagram for the runTests method

4.3.3 Some notes about XSLT

XSLT processing is fulfilled through the use of native Java interfaces (i.e. javax.xml.transforms.*), but Saxon library is used as implementation. Two reasons for choosing Saxon is the ability to:

- compile the stylesheets in order to improve the performance.
- obtain the line of given node in a DOM tree. This is useful to provide feedback to the user about the origin or cause of an error. These lines reference a location inside the moki document.

In general XSLT tests use XPath expressions for accessing the relevant parts in moki document. The templates also produce a node result for easy building of the Java Test Result object.

4.4 JUnit Tests

The W3C mobileOK Checker has been developed using the JUnit Tests Framework. The overall design/behaviour of the test cases for the mobileOK 1.0 Tests Checker library is the following:

- The input for the tests is a Web resource with additional images and stylesheets. On top of that, it is needed to provide the expected moki and results XML documents.

- These resources are exposed locally through an embedded Tomcat. One given test case will use this local host as the URI parameter to the checker library. Once the checker library is executed, the produced moki and results documents are compared to the expected ones. If both archives are identical, the test case will succesfully pass. Otherwise, the differences will be reported and the test will fail.

4.4.1 Executing JUnit tests

This section explains how to launch the JUnit tests created in the project, included in the source code in the CVS.

From Eclipse IDE

Examples are made following the steps needed in Eclipse IDE in order to execute the tests.

- Launch a new run configuration (Run option in Run menu).
- Introduce as main class `org.w3c.mwi.mobileok.basic.OneTestTest`

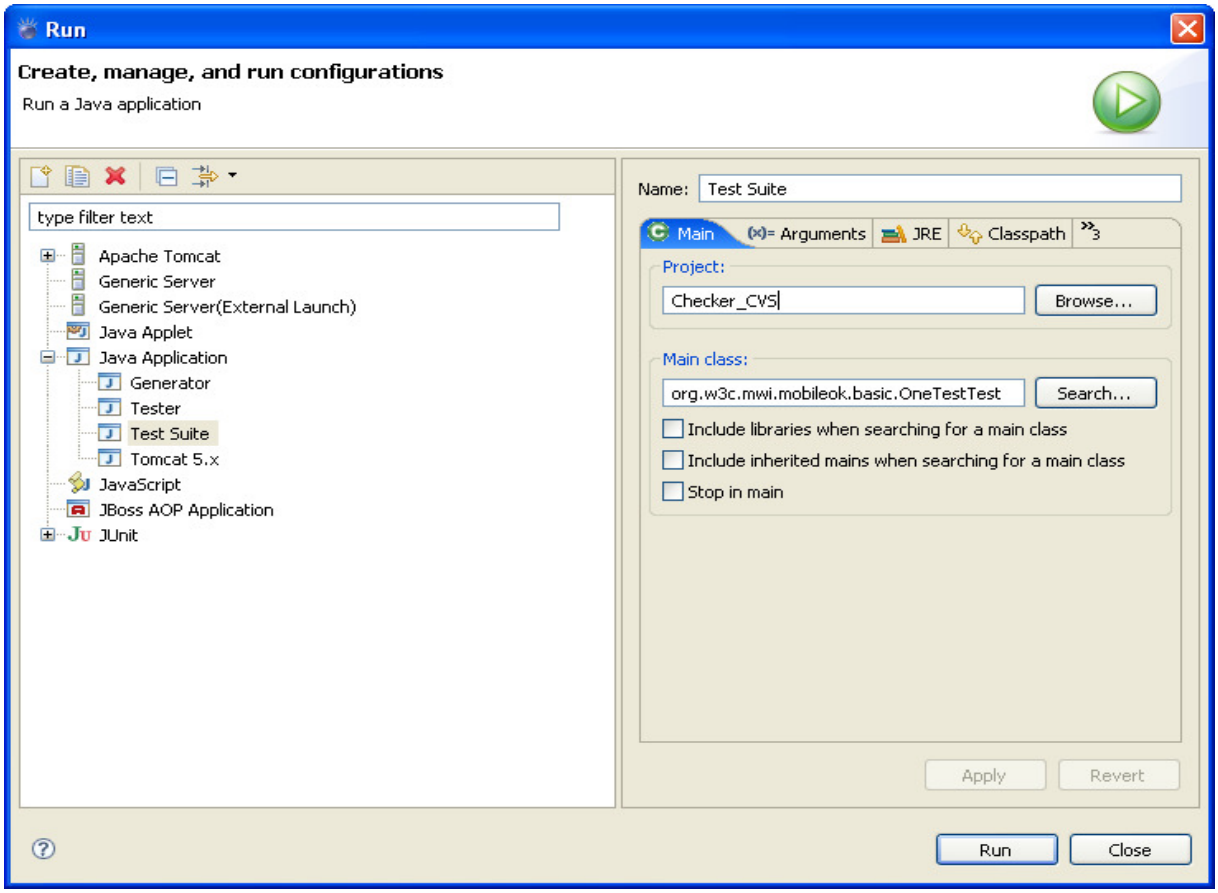

Illustration 10: Execution of a JUnit test: main class

- Enter, as program arguments, one of the mobileOK tests in the format specified in the section 3.1.2 of the mobileOK Checker 1.0b User Manual. The JUnit test for the specified mobileOK test will be launched. If no mobileOK test is specified, all tests will be launched. It is possible to execute a specific subtest for a given test. For example, in the following illustration, the execution of subtest AUTO_REFRESH-1 is showed.

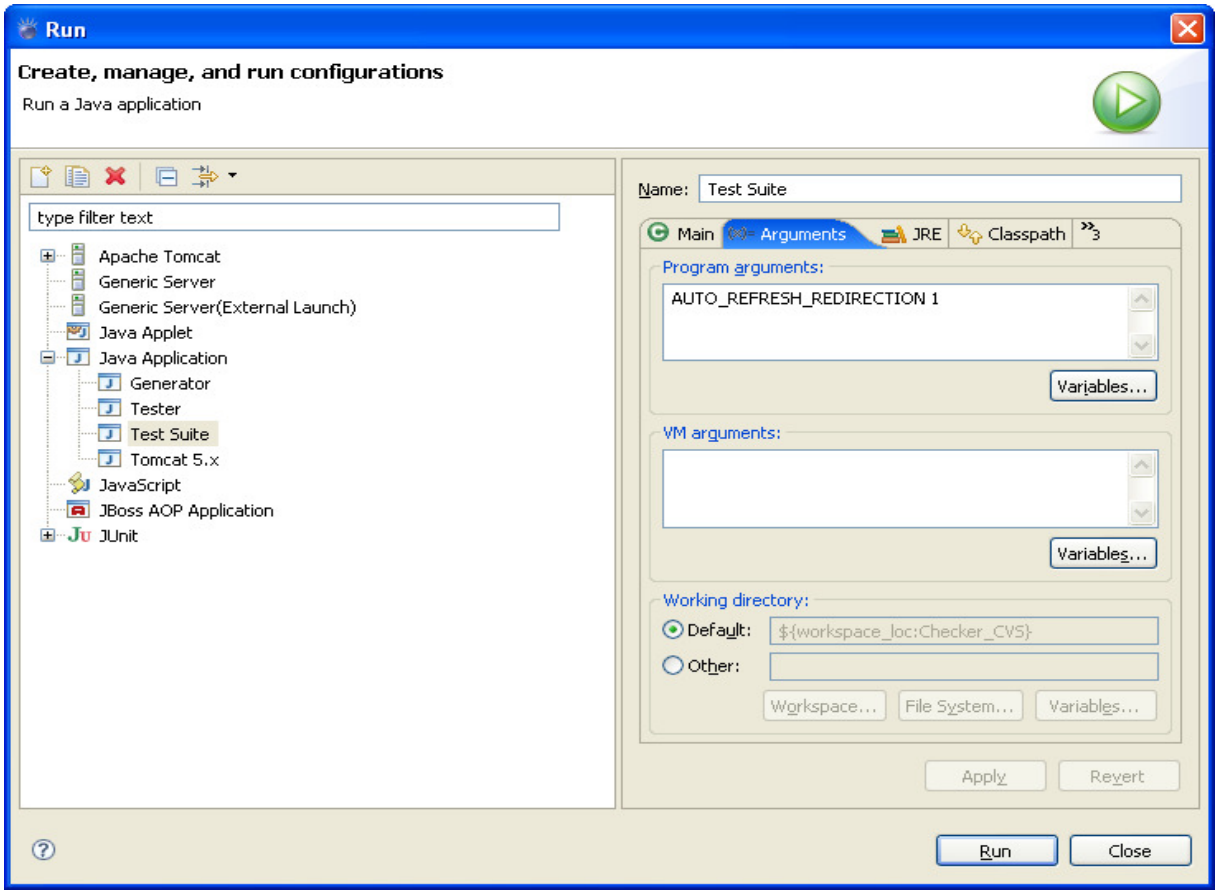

Illustration 11: Execution of a JUnit test (program arguments)

From Ant Tool

It is possible to use the Ant tool for executing JUnit tests. From the project root directory (where is located the build.xml file, which is necessary for Ant tool to run), simply type “ant test”. This will launch an Ant task, called “test”, that will execute all tests.

4.4.2 Notes on how the tests are written

Test cases are located under the "tests/data" directory in the project. There is a directory for each mobileOK test (e.g. PageSizeTest) and a subdirectory for each test case (e.g. "1", "2", "3", etc.). Note that creating a test is just a matter of creating another such subdirectory.

Within each directory, the following archives are found:

- index.xhtml: the main document to be tested.
- moki.xml : the expected moki document output from the checker.
- testresults.xml : the expected test results document

It is also possible to include other files, such as stylesheets and images. They will all be served to the checker as expected. (i.e, it is possible to include as much images or stylesheets referenced from index.xhtml as needed)

Note that for each file 'foo' it is possible to include a companion file called 'foo.headers' which lists the HTTP headers which should be sent when serving that file. This is needed for tests that need to check for particular header values. These are just text files with one header name/value pair per line, such us:

- Content-Type: application/xhtml+xml
- Cache-Control: public
- ...

The execution of a given test will produce, at the end, the comparison between the expected moki.xml and testresult.xml and the real ones obtained. For that reason, the possible erros are reported in terms of the differences between both kind of documents.

4.5 Implementation Decisions

All classes are in the same package *org.w3c.mwi.mobileok.basic*. On the other hand, most methods have visibility at package level. This decision -together with the fact that most classes are final- is due to the fact that the project has very specific requirements and it was not designed to be extended at code level.

Additionally, a third party could create a software tool that might process the moki document for its own purposes, as that document contains a lot of useful information valuable for an external piece of software.

4.6 Third-parties software

This section is just a reminder of the most important third parties used inside the library to fullfill tasks such as image or markup validation among others.

- CSS Validator: This library is used to validate CSS stuff.
- HTTPClient: Library for management of HTTP connections.
- JHove: Used for image (JPEG or GIF) validation.
- TagSoup: This tidying tool is used to clean the markup before building the DOM tree.
- Saxon: This XSLT processor is used for two purposes. First of all, this library provides a method to obtain the line of given node in a DOM tree. On the other hand, it has a feature to compile the stylesheets being used in order to improve the performance.

5 Cookbook

This section introduces some common usages of the library.

5.1 Basic Java Usage

This first code snippet represents the launching of all the tests against a given site.Once the results are obtained, it would be possible to process the results DOM tree, for example in order to print it to the common output.

```
URI uri=new URI("http://www.mymobilesite.com");
Tester tester = new Tester(uri);
TestType[] testTypes;
//all tests will be passed
testTypes = TestType.values();
TestResults results = tester.runTests(testTypes);
//Process the Result DOM. For example printing it to the common output
printResults(results.getTestResultsDOM());
```

5.2 Setting Additional Headers

This example sets specific HTTP request headers. Although it is possible to define every header of the request, it is important to be aware that if the Default Delivery Context related headers are overriden the results will not be mobileOK Basic compliant.

```
// Setting an additional value for an HTTP Accept-language header. It is possible to
// define values for any header in an HTTP request.
// Be aware that if Default Delivery Context related headers are overriden by this
// configuration, the results will not be valid from the point of view of the
// mobileOK Basic Tests
Map<String,String>    myAdditionalHeaders    =    new    HashMap<String,String>();
myAdditionalHeaders.put("Accept-Language", "ja");
Map<String,Object> configMap = new HashMap<String,Object>();
configMap.put(TesterConfiguration.HTTP_REQUEST_HEADER_OVERRIDES_KEY,
myAdditionalHeaders);
TesterConfiguration config = new TesterConfiguration(configMap);
TesterConfiguration.setConfiguration(config);
//...Usual basic usage like previous example
```

5.3 Using moki document for other purposes

In the follwing example a possible usage of the moki document for other purposes than the mobileOK Basic Tests is introduced. This sample is about a hypothetical scenario in which a third party is interested in knowing if a given Web page has broken links or not.

Once the moki document is obtained, a suitable XPath expression for locating broken links is needed (This example uses XPath expressions from Java. It would be possible to use a declarative approach such as a XSLT based one)

Note how this is fullfilled through the access to the HTTPResponse blocks of each link in the moki document. On top of that, the last sucessful response is needed (in order to avoid possible redirections). Finally, there is a status header which keeps the code of the responder and it is exactly what is needed.

```
URI uri=new URI("http://www.mymobilesite.org");
Tester tester = new Tester(uri);
//Obtaining the moki document
Document moki = tester.getPreprocessorResults().getMokiDocument();
//Proper initialization of XPATH expressions
XPathFactory factory = XPathFactory.newInstance();
XPath xpath = factory.newXPath();
//Fix an appropiate namespaces for the moki
xpath.setNamespaceContext(new MokiNamespaceContext());
XPathExpression BROKEN_LINKS;
//A suitable XPath expression is needed
BROKEN_LINKS = xpath.compile("/moki:moki/moki:links/moki:link/" +
                "moki:retrieval/moki:HTTPResponse[last()]/moki:status[@code!='200']");
Boolean result=(Boolean)BROKEN_LINKS.evaluate(moki, XpathConstants.BOOLEAN);
```

One important issue is to properly set the namespaces of the moki to the XPath expression. The following class takes care of that.

```
public class MokiNamespaceContext implements NamespaceContext {
    private  Map<String, String> prefixMap;
        public MokiNamespaceContext() {
            this.prefixMap = new HashMap<String, String>();
            prefixMap.put("moki", "http://www.w3.org/2007/05/moki");
            prefixMap.put("xml", XMLConstants.XML_NS_URI);
        }
        public String getNamespaceURI(final String prefix) {

            if (prefixMap.containsKey(prefix)) {
                    return prefixMap.get(prefix);
            } else {
                    return XMLConstants.NULL_NS_URI;
            }
        }
        // There is no need for implementing other methods of interface

}
```

6 Extending the library

As it was pointed out in Implementation Decisions, it is possible to extend the library using the intermediate document. It contains a lot of information about a Web Resource in general, and a new project could be built on top of it. For more information about the moki document formar, please see the mobileOK Checker 1.0b User Manual.

7 Identified improvements and extensions

7.1 CSS in XML

One initial requirement was having CSS serialized inside moki document. But, due to time constraints and not being a trivial task, this requirement was not accomplished.

However, a project for this purpose was begun with the effort of some members of the checker task force. The project, called Representation of CSS in XML, can be found in W3C CVS Repository. It is experimental code and has several things to do. Everyone interested in it can contribute to its refinement.

7.2 Encapsulating third-parties software in moki

Moki intermediate document has several parts with information messages that come from third-parties software. These parts are:

- Tidy tool output.
- XHTML Grammar validity.
- CSS Grammar validity.
- Images validity.

It is desirable to have intermediate document messages not coupled to specific third-parties software. This approach would be implemented by having moki proper message codes (or better groups of them) and mappings between third-parties' and mobileOK Tests Checker's. This way, it would be easier to replace a tool or to implement the internationalization of messages.

However, this approach is not easily applicable as none of the tools being used has a message management that satisfies the needs just commented. Two possible solutions were analyzed:

1. Modify the tools in order to fullfill messages codes (and internationalization and so on -The next option is implicit-).

2. Modify the tools in order to make them internationalizables (using properties files) and in this way make it possible to select a language for moki (e.g we could have an Spanish moki with messages of third parties in Spanish, and so on).

Perhaps a better solution would be a combination of both of them. Note that, in some cases, it is not even possible to include any kind of messages.

As the messages handling in tools are very heterogeneous, a reasonable solution would be the treatment of each tool in a separate way. This would not be the best solution in all cases but a tradeoff between development agility and code quality.

More information on this can be found in the document ThirdParties.

8 Known issues

8.1 About W3C mobileOK Checker 1.0b

The development of the tool is alive. Please do not hesitate to send your comments to the mobileOK Checker Task Force mailing list (public-mobileok-checker@w3.org). Your feedback is very important for the authors of W3C mobileOK Checker 1.0b so this software can be improved. Optionally, you can access the bug tracker of the Checker at http://www.w3.org/Bugs/Public/ and help W3C developing this tool.

8.2 About this Developer Manual

- **General issues:** Review and improve wording all across the document.

9 References

NOTE: When referring a W3C document, the version of the document that has been considered at this version of the W3C mobileOK Checker User Manual appers in this section. Additionally, the URI to the latest version of such document is offered between square brackets.

(1) W3C mobileOK Basic Tests 1.0, http://www.w3.org/TR/2007/WD-mobileOK-basic10-tests-20070928/. W3C Candidate Recommendation 30 November 2007. Editors: Sean Owen (Google) and Jo Rabin (dotMobi) [latest version: http://www.w3.org/TR/mobileOK-basic10-tests/].

(2) W3C Mobile Web Best Practices (Basic Guidelines) 1.0, http://www.w3.org/TR/2006/PR-mobile-bp-20061102/. W3C Proposed Recommendation 2 November 2006. Editors: Jo Rabin (dotMobi) and Charles McCathieNevile -early drafts- (Opera Software) [latest version: http://www.w3.org/TR/mobile-bp/].

(3) W3C License, http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231.

(4) W3C Mobile Web Best Practices working group, http://www.w3.org/2005/MWI/BPWG/.

(5) W3C Mobile Web Initiative, http://www.w3.org/Mobile/.

(6) XML Schema for the mOKI document format, http://dev.w3.org/cvsweb/2007/mobileok-ref/moki/schema/moki.xsd. Editor: Jo Rabin (dotMobi). HTML version available at http://dev.w3.org/cvsweb/2007/mobileok-ref/moki/schema/moki.xsd.html.

(7) Representation of CSS in XML, http://dev.w3.org/cvsweb/2007/cssxml/. Editor: Jo Rabin (dotMobi).

(8) mobileOK Checker 1.0b User Manual, http://dev.w3.org/cvsweb/2007/mobileok-ref/mobileOK-Basic-RI-1.0-UserManual.pdf [latest version]. Editors: Ignacio Marín and Abel Rionda (Fundación CTIC). Version available at release time of this document: http://dev.w3.org/cvsweb/2007/mobileok-ref/mobileOK-Basic-RI-1.0-UserManual.pdf?rev=1.4&content-type=text/x-cvsweb-markup.

(9) CSS Validator, http://jigsaw.w3.org/css-validator/DOWNLOAD.html. World Wide Web Consortium.

(10) CSS, Cascading Style Sheets, level 1, http://www.w3.org/TR/REC-CSS1. W3C Recommendation 17 Dec 1996 (revised 11 Jan 1999). Editors: Håkon Wium Lie and Bert Bos.

(11) JUnit Tests Framework, http://www.junit.org/.

(12) Ant, http://ant.apache.org/. The Apache Software Foundation.

(13) HTTPClient, http://hc.apache.org/httpclient-3.x/. The Apache Software Foundation.

(14)JHove (JSTOR/Harvard Object Validation Environment), http://hul.harvard.edu/jhove/. Copyright 2003-2007 by JSTOR and the President and Fellows of Harvard College.

(15)TagSoup, http://ccil.org/~cowan/XML/tagsoup/.

(16)Saxon, http://saxon.sourceforge.net/. The XSLT and XQuery Processor.

(17)ThirdParties, http://docs.google.com/Doc?docid=dhbw7zt7_0f8w6bq. A study of Third-Parties software considered useful to be reused in the implementation of the mobileOK Tests 1.0 Checker. Editors: Miguel García and Abel Rionda (Fundación CTIC).

(18)W3C CVS Repository for mobileOK Checker, http://dev.w3.org/cvsweb/2007/mobileok-ref/

10 Acknowledgements

W3C mobileOK Checker 1.0b is the result of the effort of several organizations. In alphabetical order, the members of the W3C mobileOK Checker Task Force (taking place within the W3C Mobile Web Best Practices working group) who contributed to the creation of this software tool and its documentation are:

- Roland Gülle (Sevenval)
- Miguel García (Fundación CTIC)
- Dominique Hazäel-Massieux (W3C)
- Laura Holmes (Google)
- Ignacio Marín (Fundación CTIC)
- Ruadhan O'Donoghue (dotMobi)
- Sean Owen (Google, Task Force leader)
- Jo Rabin (dotMobi)
- Abel Rionda (Fundación CTIC)